REMARKS

Claims 1-13 are currently pending in the application. By this amendment, claims 1-13 are amended for the Examiner's consideration. The foregoing separate sheets marked as "Listing of Claims" shows all the claims in the application, with an indication of the current status of each. Support for the amendments to the claims is found in the specification and drawings, and in particular at page 2, lines 9-11; page 6, line 8 to page 12, line 20; and drawing figures 1A, 1H, 1AA, and 3-9.

In the specification, the paragraphs beginning at page 2, line 6; page 7, line 11; page 9, line 1; page 10, line 1; page 12, line 1; and page 24, line 1, have been amended to correct errors in grammar and syntax and to conform the text to the drawing figures. At page 10, line 2, the reference figure for SQL databases 160 now conforms to page 10, lines 5-6 and as shown in Figures 1H and 2. Several acronyms well known to those skilled in the art are provided with explicit translations. At page 8, line 3, a definition ("Java ARchive") is provided for the acronym well known to those skilled in the art as "Jar." At page 10, line 14, a definition ("Java 2 Platform, Enterprise Edition") is provided for the acronym well known to those skilled in the art as "J2EE." At page 24, line 3, a definition ("Java Naming and Directory Interface") is provided for the acronym well known to those skilled in the art as "JNDI." No new matter has been added.

The Examiner has required a replacement oath or declaration acknowledging the duty to disclose under 37 CFR §1.56, indicating that a signature over a complete post office address is missing from the application papers. A copy of the Oath and Declaration, including a second page containing those elements apparently missing from the file available to the Examiner, which was submitted with the application, is attached to this amendment. It is believed, therefore, that the Examiner's requirement is satisfied.

The Examiner has rejected claims 1-13 under 35 U.S.C. §112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which the applicant regards as the invention. In particular the Examiner indicates that claim 1 is incomplete because it omits essential elements, i.e. fails to claim that the system is automatic or where the conversion of the database information is occurring. The Examiner similarly rejects claims 4-13 for failing to indicate how assistance is provided to the overall method for generating database application conversion automatically. The claims have been amended to overcome these grounds of rejection.

The Examiner has rejected claims 1-13 under 35 U.S.C. §102(a) as being anticipated by U.S. Patent No. 6,591,272 to Williams. Williams appears to be a method for a) reading the definitional elements of a database to determine data types and interrelationships between relational data elements, b) assembling them into a vendor neutral standardized view of the database schemas, and c) creating all the possible logical objects that are in the database. A user would then select a subset of these objects for an application. The problem being addressed by Williams is "the inherent mismatch between data stored in relational databases and the format and structure of this relational data in object based systems" (col. 3, lines 25-28). Usually, it takes an expert of considerable skill to manually map database tuples into objects, even using object-relational mapping tools. Under the solution provided by Williams, this mapping can now be done by an inexperienced user.

Before considering the differences between Williams and the present invention, it is helpful to consider what the two inventions have in common: both are methods of converting databases to a standardized format. In Williams, the conversion is from a specific query language (SQL) database to a standardized format, which serves to facilitate ultimate conversion at the client side to a desired format. The standardized format is not itself the end product of the conversion desired by the user. The purpose of the standardized format is to enable

inexperienced users to obtain appropriate objects for use in an application at a client. Creation of these usable objects is done at runtime, using source code (which has been merged into the standardized view) to encapsulate the metadata and data values of the object desired by the user. The client side user is then able to assemble the object into the format required by his application. It should be noted that the objects are source code segments composed of functionality and data. It should further be noted that Williams adheres to the Common Object Request Broker Architecture (CORBA), which is supported by a consortium known as the Object Management Group and whose Interface Definition Language (IDL) is used by clients to invoke an operation on an object on a remote server. IDL is independent of programmng language, but maps to all popular programming languages (e.g. C, C++, Java, COBOL, Smalltalk, Ada, Lisp, etc.).

By contrast, the present invention does not use an intermediate standardized format. Although a Standard Query Language (SQL) example is described in detail in the specification, the invention is not limited to that example. However, according to the methodology of the invention, an equivalent to the "bean grinder" would be developed for different database examples. In any case, the target for the conversion is the Enterprise Java model, and the described and claimed methodology is usable to convert any database to the Enterprise Java model. In contrast to Williams, the purpose of the present invention is to avoid the labor intensive exercise which even experienced software engineers must undergo under the prior art to convert a database to the Java model. The methodology for accomplishing this purpose has several important characteristics. First, it reverses the conventional method of conversion by working from the database specification to the target Java model. The conventional approach would be for the application developer – and it is important to emphasize that the "user" of the present invention is the developer of an Enterprise Java application – to first write the application in Java, and then determine how the database can be mapped to the target Enterprise Java model. It should be noted that

Williams is similar in this respect, in that it begins with the database to be converted. However, Williams has no alternative since the target database is not known. Instead, Williams converts a SQL database to an intermediate standardized form. The structure of the standardized form (metadata coupled with skeleton code templates representative of final classes) avoids the requirement that the user have a high level of expertise in order to deal with the "inherent mismatch between data stored in relational databases and the format and structure of this relational data in object based systems" (see above citation).

Thus the appropriate analogy is between the standardized form of Williams and the Enterprise Java model of the present invention. In Williams, the direction is from SQL to many different databases, via the standardized form. In the present invention, the direction is from any database to the Java model. While it is conceivable that teachings applicable to a one-to-many conversion problem may anticipate a methodology responsive to a many-to-one conversion problem, such a conclusion is not justified merely by the common use of "SQL" in both Williams and the present invention and the fact that Enterprise Java is an object oriented model of the kind toward which the standardized form of Williams is directed. As will be explained hereafter, the methodologies are different. The foregoing discussion concerning the differences between the problems addressed and the objectives sought by Williams and the present invention serves to emphasize why the methodologies are not the same and why the teachings of Williams do not anticipate the methodology undertaken by the present invention.

The methodology of the present invention begins with the recognition that simply "automating" the laborious and complex task presented by the conventional method is not adequate. For even one table in a database, converting that table into usable source code in Enterprise Java's object oriented approach to providing functionality and data to remote users over the Internet requires that "numerous files" of Enterprise Java source code be generated (page 6, line 19). In the

conventional approach, the developer then uses the Enterprise Java deployment descriptor file to back his way into the name assignments necessary to coordinate the database table among the "numerous files" in the Enterprise Java model. The problem is that this is an error prone process, and furthermore must be reviewed each time there is a change in the database itself. The insight of the present invention is that it makes more sense – however counterintuitive – to begin with the database tables and generate from the database the "numerous files" required by the Enterprise Java model at the beginning, before the developer writes code for the Enterprise Java application. This reversal of order is what is referred to in the specification (page 6, lines 11-12) as "reverses the conventional method of conversion by wlorking from the database specification to the EJB".

It turns out that the thorough and methodical use of metadata in the source database, as described in great detail in Figures 3-9 and accompanying text, is both necessary and sufficient to automatically accomplish the coordination of names and functions among the "numerous files" of the Enterprise Java model in order to make a particular database table available for use in the Enterprise Java application being developed by the developers. The apparently rote nature of what is described in the application is misleading. A more selective approach – simply "automating" what is typically undertaken by the conventional method – results in source code whose viability cannot be tested until runtime, that is, until after the developed application is deployed (see page 2, lines 10-12). This is not an acceptable situation, and is the reason why the reverse approach of the present invention was pursued. It should be emphasized again that the approach of the present invention is to proceed directly and without an intermediate structure (as the "standardized format" of Williams) to source code usable at run-time. By generating the plurality of Enterprise Java files necessary for a database table to be usable in an Enterprise Java application, and doing so from the database so that these files can be used as components in the application which is then developed, the necessary coordination between table names and function names

among these plurality of files is thereby assured. It is not self-evident why this should be so, but it is.

The test of this proposition is its use during development of the Enterprise Java application. Whenever the source database is changed, the automated method (which is implemented as a "bean grinder") is simply repeated and the "numerous files" are produced again, with the confidence that the table names and function names will be consistent across these "numerous files." The process is linear and direct, with all necessary information being gathered at the outset, thereby promoting a high-speed automated solution. It is important to note that the method of the invention – implemented for the Enterprise Java model as the "bean grinder" – results in actual EJB source code components, rather than going through an intermediate structure as in Williams.

The significance of coordination between table references – table names and the names of functions providing desired functionality – across the plurality of files may be seen from the specification. For example, the "table name" appears across these files (see page 13, line 3, for DATA file; page 14, line 10, for HOME file; page 15, lines 18-19, for Remote file; page 17, lines 1-2, for Bean file; page 20, line 4, for PrimaryKey file; page 21, line 14, for Persistent file; page 27, line 2, for methods listed in the Deployment Descriptor file; and page 27, line 11, for container managed transactions listed in the Deployment Descriptor file for methods; and page 29, line 7, for the file specifying the target application server). Similarly, the functionality embodied in methods also appears in more than one file. For example, the "create" function appears in both the home interface file (page 12, line 13) and the persistent file (page 12, line 15).

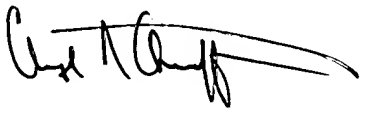
The claims have been amended to clarify and emphasize the foregoing aspects of the invention that are clearly distinct from Williams.

In view of the foregoing, it is requested that the application be reconsidered, that claims 1-13 be allowed, and that the application be passed to issue.

Should the Examiner find the application to be other than in condition for allowance, the Examiner is requested to contact the undersigned at 703-787-9400 (fax: 703-787-7557; email: clyde@wcc-ip.com) to discuss any other changes deemed necessary in a telephonic or personal interview.

If an extension of time is required for this response to be considered as being timely filed, a conditional petition is hereby made for such extension of time. Please charge any deficiencies in fees and credit any overpayment of fees to Attorney's Deposit Account No. 50-2041.

Respectfully submitted,



Clyde R Christofferson
Reg. No. 34,138

Whitham, Curtis & Christofferson, P.C.
11491 Sunset Hills Road, Suite 340
Reston, VA 20190
703-787-9400
703-787-7557 (fax)

Customer No. 30743